

Application of Program Slicing			
Introduction	Program Slicing	Reverse Engineering	Maintaining

Application of Program Slicing

Selection
Dynamic Slicing
Static Slicing
Program Slicing
Program Slicing

A dynamic slice may be used to identify the parts of the program that contribute to the computation of the selected function for a given program execution (program input). Dynamic slicing may help to narrow down the program part that contributes to the computation of the function of interest for a particular program input. A dynamic slice is usually represented in the textual form, i.e., a dynamic slice is displayed to programmers in the form of highlighted statements in the original program or in the form of a subprogram by removing from the original program all statements that do not belong to the slice. Although dynamic slicing does reduce the size of a slice, it is still not the programmer to analyze the dynamic slice. For some programs, there may be a relatively small decrease in the size of a slice, or because of the significant size of the software system, the size of a slice may still be very large and hard to comprehend. Programmers may still have difficulty understanding the program and its behavior. Since dynamic slices are only represented in the textual form, they may provide limited guidance in the process of understanding program execution. Therefore, it is important to devise methods that concentrate the programmer's attention only on the parts of the program and its execution that relate to the computation of the function of interest. It has been shown that dynamic program slicing is not only useful in software debugging but also in software maintenance, program comprehension, and software testing.

Traditionally, in order to understand a program execution, a programmer uses conventional debuggers that support breakpoint facilities and stepwise program execution. Breakpoints allow a programmer to specify places in a program where the execution should be suspended. When a breakpoint is reached and the execution is suspended, the programmer can then examine various components of the program state and check the values of program variables. Programmers may also execute a program in a step-wise manner in order to observe the program execution. Conventional debuggers, however, do not provide any means to identify those program parts whose execution contributes to the computation of the function of interest. Using debuggers is an inefficient and time-consuming approach for understanding program execution, especially when a programmer is interested in understanding those parts of the program execution that relate to a particular program function. The programmer may observe a large amount of unrelated computation and it is frequently almost impossible for him/her to distinguish relevant computations from non-related computations. In order to make the process of program understanding more efficient, it is important to reduce the amount of information a programmer sees and to focus his/her attention on related computations. Dynamic slicing techniques provide a means to prune away unrelated computation. In dynamic slice computation, different types of information are computed. For example, contributing actions and non-contributing actions are computed. This information is usually discarded after the slice computation. Program comprehension, as part of the software maintenance process, is a crucial phase of system development. It is expected that a major share of systems development effort goes into modifying and extending preexisting systems about which we usually know little [4]. Through the increased complexity of software systems, their maintenance is becoming more and more aggravating. Change to a system may be necessitated for adaptive, perfective, corrective, or preventive reasons. Understanding the system, incorporating the change, and testing the system to ensure the change had no unintended effect on the system are the three facets of software maintenance. Reverse engineering of program comprehension can be described as analyzing a subject system to (a) identify the system's components and their interrelationships, (b) to create representations of a system in another form at a higher level of abstraction, and (c) to understand the program execution and the sequence in which it occurred.

Best Available Copy

Dynamic Slicing			
Selection	Path-sensitive slicing	Abstract interpretation	Summary

Dynamic Slicing

Selection
Selection
Selection
Selection
Selection

A dynamic program slice is this part of a program that "affects" the computation of a variable of interest during program execution on a specific program input. Dynamic program slicing refers to a collection of program methods that are based on program execution and may significantly reduce the size of a program slice because run-time information, collected during program execution, is used to compute program slices. Dynamic program slicing was originally proposed only for program debugging, but its application has been extended to program comprehension, software testing, and software maintenance. Different types of dynamic program slices, together with algorithms to compute them, have been proposed in the literature.

A static program slice consists of all statements in a program that may affect the value of variable v at some point p . As originally introduced, static slicing involves all possible program executions, i.e., the static slice preserves the computation of variable v at point p for all program inputs. However, in many situations one is interested in a slice that preserves the program's behavior for a specific program input, rather than that for all inputs. This type of slice is referred to as *dynamic* slicing. The concept of dynamic program slicing was for the first time presented in [1]. Taking a particular program execution into account, dynamic slicing may significantly reduce the size of the slice compared to static slicing. During program execution it is possible to monitor different types of run-time information that can be used during dynamic slice computation, e.g., addresses of array elements, addresses of elements of dynamic data structures that are referenced during program execution, etc. Most of the existing dynamic slicing techniques have been proposed for sequential programs. But, the concept of dynamic slicing has been extended to distributed programs. Two major types of dynamic slicing have been proposed in the literature: executable dynamic slicing and non-executable dynamic slicing. Informally, an executable dynamic slice is a subprogram that can be executed and it preserves a value of a variable of interest. On the other hand, a non-executable slice is a subprogram that contains statements that "influence" the variable of interest and, in most cases, it cannot be executed.

A dynamic slice on the other hand overcomes the limitations of the static algorithms by being based on a particular program execution (program input). Therefore the dynamic slicing computation can utilize information about the actual program flow for a particular program execution, which leads to an accurate handling of dynamic and conditional language constructs and therefore to smaller program slices. One of the major drawbacks of dynamic slicing (compared with static slicing) is that it is necessary to identify relevant input conditions for which a slice should be computed. A commonly used approach to identify such input conditions is referred to as an operational profile. It is a well-known concept that is frequently applied in testing and software quality assurance. For the determination of the profile the relative frequencies of occurrence of the run types (inputs) are used, expressed as fractions of the total of runs representing their probabilities.

Best Available Copy

Static Slicing

Static Slicing

Selection

Based on the original definition of Weiser [Wei84] a static program slice S consists of all statements in P that may affect the value of variable v at some point p . The slice is defined for a slicing criterion $C=(x, V)$, v a statement in program P and V is a subset of variables in P . Given C , the slice consists of all statements in P that potentially affect variables in V at position x . A static slicing includes all the statements that affect variable v for a set of *all* possible inputs at the point of interest. Static slices are computed by finding consecutive sets of relevant statements, according to data and control dependencies.

The program dependence graph (PDG) was originally defined by Ottenstein and Ottenstein [Ott84] and later by Horwitz et al.[Hor88, Hor90, Rep88, and Rep89]. Data and control dependencies between nodes may be used to form a program dependence graph, that can be used for the computation of static slices by traversing backwards along edges of the program dependence graph from a point of interest. The program dependence graph is formed by combining all data and control dependencies that exist in the program. The static slice of a program with respect to a variable v at a node i , consists of all nodes whose execution could possibly affect the value of the variable v at node i . The static slice can be easily constructed with the PDG by traversing backwards along the edges of the program dependence graph starting at a node i . The nodes, which were visited during the traversal, constitute the desired slice [Ott84].

The strength of static slicing lies in particular in the following areas: (a) the computation of a static slice is cheap (compared to the dynamic slice) hence only the static analysis of the source code and no program execution is required, (b) it is useful if a user wants to gain a general understanding of the program parts that contribute to the computation of a selected function with respect to all possible program executions

Best Available Copy